NuMI Horn Longitudinal Field-Mapping System

Adam Watts

TSD Topical Meeting

19 September 2019

Motivation

For quality control, new NuMI horns go through a field-mapping process to ensure required field quality while the horn pulses in the MI-8 test cage.

This talk focuses on the mapping system for the field along the center axis of the horn, where a 3-D hall probe is scanned longitudinally through the pulsing horn to measure what should be a flat and very low magnetic field.

The original implementation of this system relied on proprietary electronics and software that have been obsoleted by the vendor, and field measurements were written down by hand during the measurement. An open-source alternative was sought to allow continued development and use of the measurement system without relying on the vendor; unification of the motor drive computer and field probe DAQ will greatly speed up the mapping process.

Original System, Mechanical

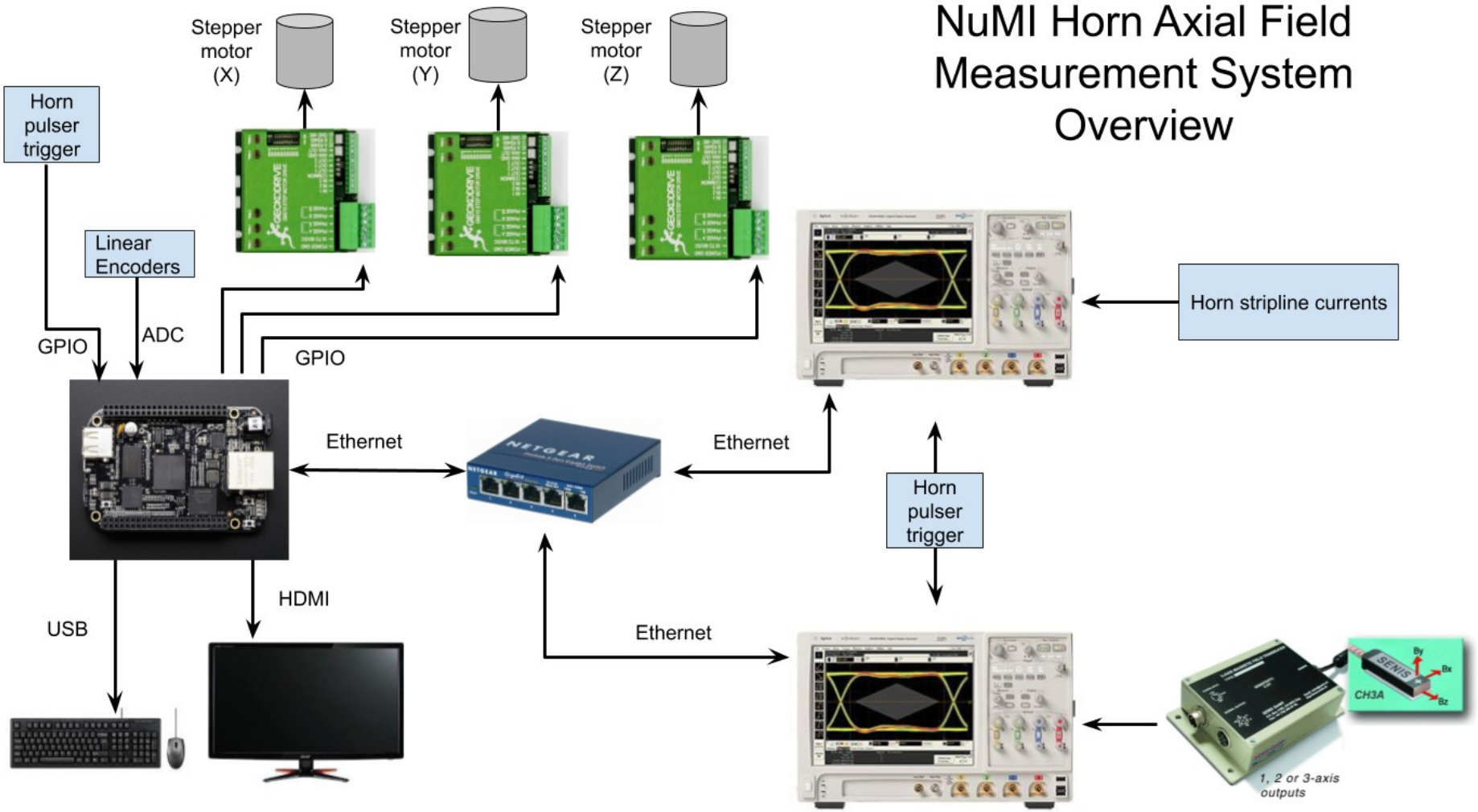
- Three-axis control
- Parker Zeta83-135-MO stepper motors
- 382 oz-in (Nm) static torque
- NEMA 34 frame
- 10.47 oz-in rotor inertia
- 3/8" shafts
- Not well documented at all. Unclear what the motor phase current rating is, though NEMA 34 standard is a clue.
- Linear encoders, one per axis. Connectors appear damaged.
- No apparent motion stops or limit switches. Unclear if motors are slip-clutch coupled.

Original System, Electronics

Stepper motor electronics:

- ZETA4 stepper motor system from Parker Motion
- Proprietary, uses “Motion Architect” software to talk to controllers
- Motion architect only runs on Windows XP or older
- Requires: legacy Windows OS, PC with serial ports or USB-to-serial adapters
- One controller module (pictured) per motorMagnetic field probes:
- Single 3D probe
- Probe reading manually recorded (i.e. paper and pencil!)

New System, Overview

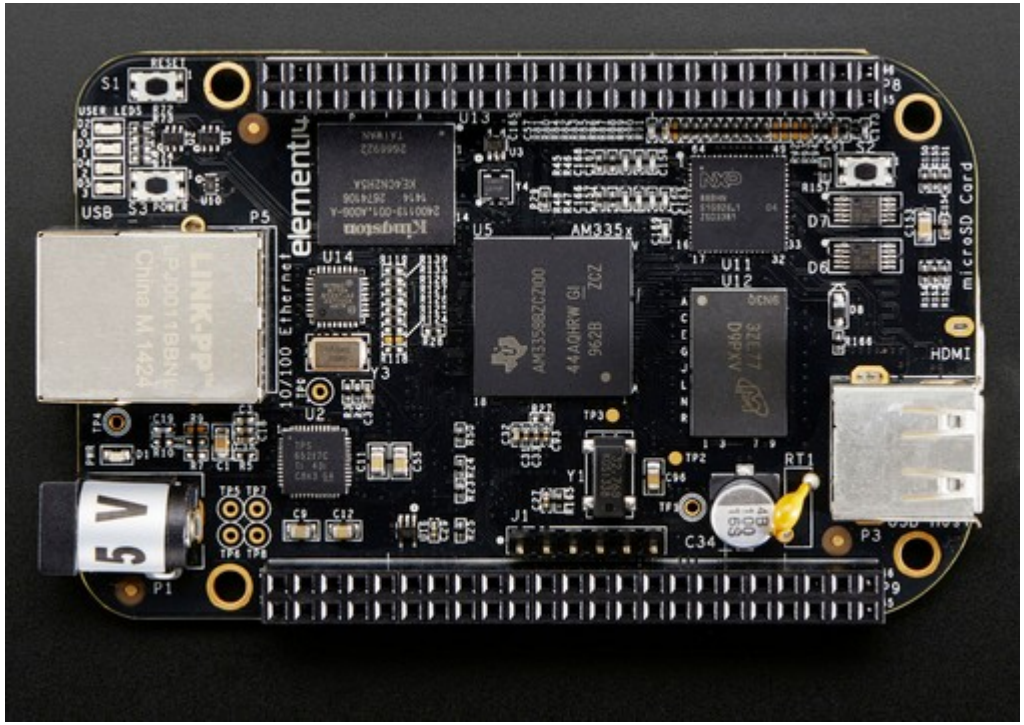


Fermilab

New System, CPU

"Beaglebone Black"
Credit-card-sized single-board Linux computer, open-source hardware with open-source software and OS

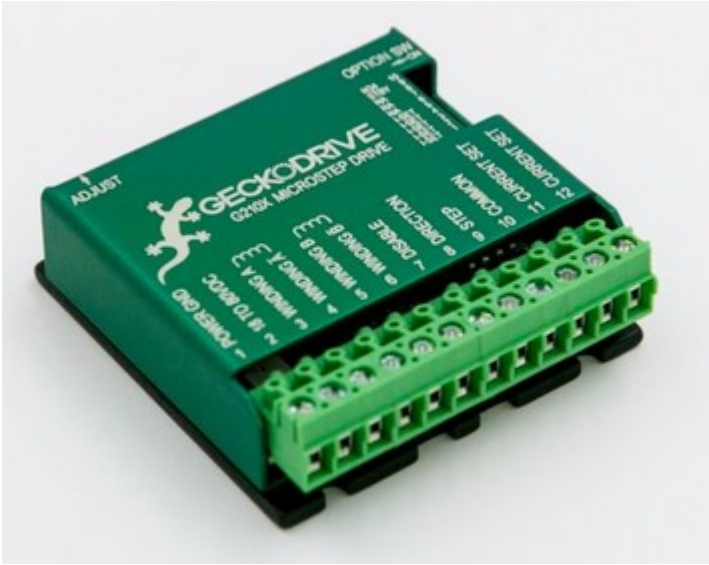
- Debian or Ubuntu Linux
- 1 GHz ARM Cortex-A8 CPU
- 512MB DDR3 RAM
- 4GB 8-bit eMMC on-board flash storage
- 2x PRU 32-bit microcontrollers
- Ethernet, micro-HDMI, USB
- 8 ADC pins, 0-1.8V, 12-bit resolution
- 62 GPIO pins
- GPIO/ADC open-source Python library freely available and supported
- External storage: microSD or USB flash



New System, Motor Controllers

Geckodrive GM210X Step Motor Drive

- 80VDC, 7A maximum output to motor, one controller per motor
- Motor power can be set with onboard DIP switches
- Built-in pulse multiplier, handles acceleration/deceleration curves, opto-isolated inputs to protect CPU board
- Driven by CPU board GPIO pin; set GPIO output pin high for ~10 milliseconds, motor controller drives motor set number of steps based on internal DIP switches
- Power supply, one per motor 36V 7.0A



Motor Motion Sample Code

- Python GPIO library for the CPU board is very straightforward
- On the right is example code that waits for user input; “s” key moves motor one way, “d” key moves motor the other way.
- Move amount per button press determined by DIP switch settings for the controller’s pulse-multiplier board. Can also modify the code to send multiple pulses per button press and speed up movement if necessary.

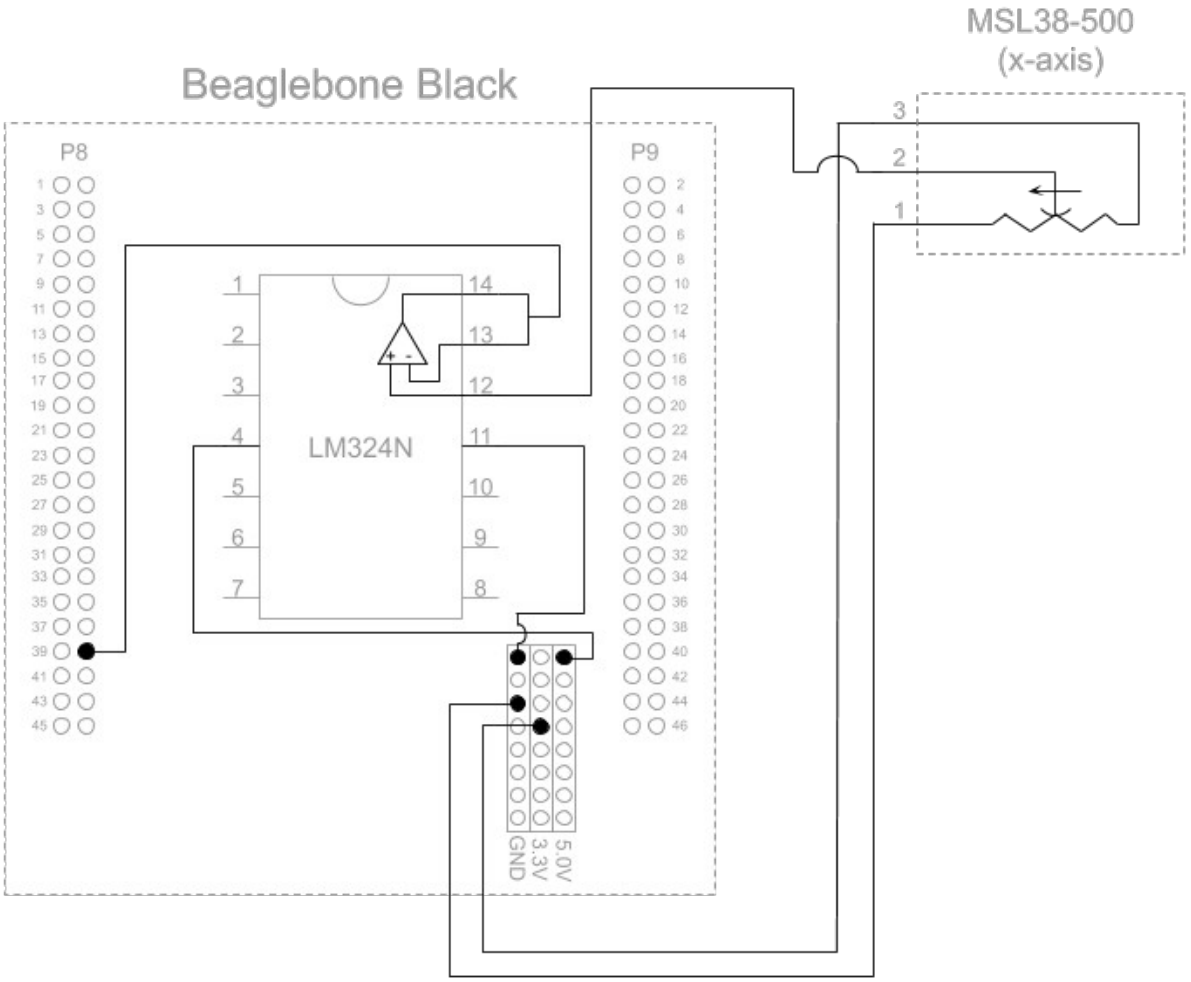
```
import Adafruit_BBIO.GPIO as GPIO

dir_pin = 'P8_8'
step_pin = 'P8_10'

GPIO.setup(dir_pin, GPIO.OUT)
GPIO.setup(step_pin, GPIO.OUT)

def moveLeft(dir_pin, step_pin):
    GPIO.output(dir_pin, GPIO.HIGH)
    GPIO.output(step_pin, GPIO.HIGH)
    time.sleep(0.001)
    GPIO.output(dir_pin, GPIO.LOW)
    GPIO.output(step_pin, GPIO.LOW)

def moveRight(dir_pin, step_pin):
    GPIO.output(dir_pin, GPIO.LOW)
    GPIO.output(step_pin, GPIO.HIGH)
    time.sleep(0.001)
    GPIO.output(dir_pin, GPIO.LOW)
    GPIO.output(step_pin, GPIO.LOW)

print 'Press: <s> = move left, <d> = move right, then <Enter>.'
while 1:
    variable = raw_input()
    if variable == 's':
        print 'Moving left'
        moveLeft(dir_pin, step_pin)
    elif variable == 'd':
        print 'Moving right'
        moveRight(dir_pin, step_pin)
```

New System, Linear Position Sensors

- Linear encoders (old system) difficult to read, as magnetic transitions must be digitized at high rate and counted.
- Simpler solution: industrial linear slide potentiometer.
- Megatron MSL38, enviromentally-sealed for industrial applications, models from 100mm to 2m effective electrical travel length.
- By applying a reference voltage and reading the divider ration from the potentiometer, the position can be measured asynchronously with the slow built-in ADCs in the CPU board.
- Linear to ±0.05% over electrical length. Resolution limited by ADC of CPU board (12-bit).

New System, Linear Position Sensors

A buffer circuit is necessary to prevent too much current drawn through the wiper, as per datasheet. Reference voltage is taken from the CPU board's 3.3V rail, which is also the reference for the ADC pins.

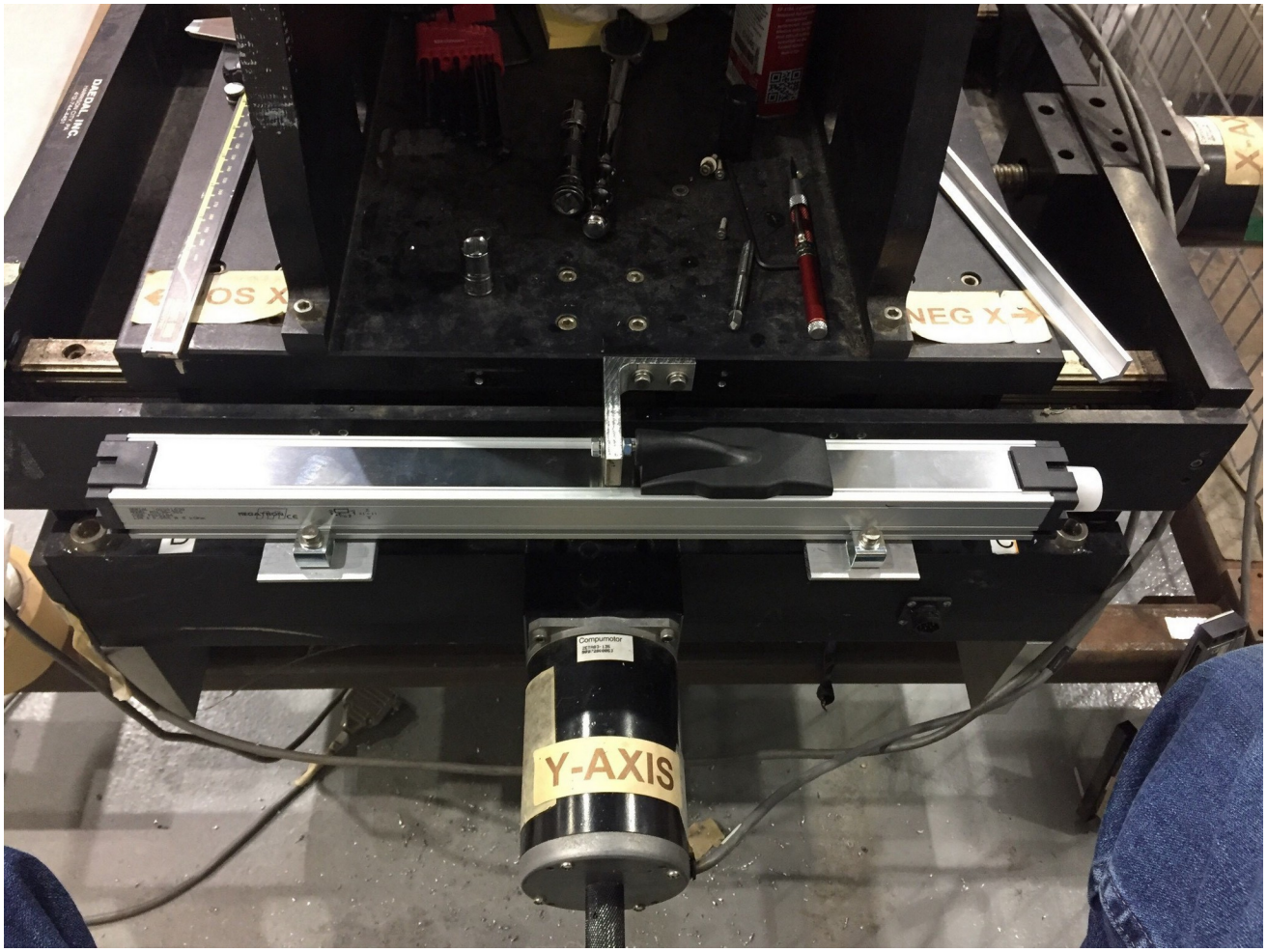


Fermilab

New System, Linear Position Sensors

Sensor mounted to axial measurement sled. Thanks Clay!

The plan is to verify this sensor is what we want to use, then purchase the other two and install them.



Fermilab

Data Acquisition

- Jim would like the 3D hall probe signals and horn stripline currents digitized @ >100 points per pulse
- Most reliable DAQ for this is an ethernet-capable oscilliscope
- Network switch creates LAN between CPU board and two scopes with static IP addresses
- Simple Python code to communicate to scopes, configure their settings, and capture data

```
import visa
import Adafruit_BBIO.GPIO as GPIO
import struct
import numpy as np

def getChannelData(ch_num):
    print "Getting data from Channel "+str(ch_num)
    scope.write('DATA:SOU CH'+str(ch_num))
    scope.write('DATA:ENC RPB')
    scope.write('WFMPre:NR_Pt 500')
    print scope.query('WFMPre:NR_Pt?')
    ymult = float(scope.query('WFMPRE:YMULT?'))
    yzero = float(scope.query('WFMPRE:YZERO?'))
    yoff = float(scope.query('WFMPRE:YOFF?'))
    xincr = float(scope.query('WFMPRE:XINCR?'))

    scope.write('CURVE?')
    data = scope.read_raw()
    headerlen = 2+int(data[1])
    header = data[:headerlen]
    ADC_wave = data[headerlen:-1]

    ADC_wave = np.array(struct.unpack('%sB' % len(ADC_wave),ADC_wave))
    voltage = (ADC_wave-yoff)*ymult+yzero
    time = np.arange(0,xincr*len(voltage),xincr)

    return time, voltage

# Initialize connection to scope
print "Connecting to scope."
rm = visa.ResourceManager('@py')
scope = rm.open_resource('TCPIP0::192.168.1.3::inst0::INSTR')
print scope.query('*IDN?')

# Initalize trigger on BBB
trigger_pin = "P8_8"
GPIO.setup(trigger_pin, GPIO.IN)

# Wait for first trigger to start data capture loop, arm scope trigger
print "Waiting for initial trigger."
GPIO.wait_for_edge(trigger_pin, GPIO.RISING)
scope.write('FPAnel:PRESS SINGLESEQ')
GPIO.wait_for_edge(trigger_pin, GPIO.RISING)
time1, voltage1 = getChannelData('1')
time2, voltage2 = getChannelData('2')
time3, voltage3 = getChannelData('3')
time4, voltage4 = getChannelData('4')
```

Progress report

Completed

- Motor movement verified with new controller and CPU board
- DAQ verified at requested rate using scopes over local Ethernet LAN, acquired and saved by CPU board
- Housekeeping: old electronics uninstalled, new monitor, keyboard/mouse, power supplies, etc. installed.
- X-axis linear sensor installed (thanks Clay!)
- Linear sensor buffer circuit built and connected to CPU board
- Sensor cables terminated, terminal block and patch panel made for PC cart so there's adequate strain relief.

To do

- Terminate new cables for motors, create patch panel for their connectors into motor controllers (extra cables for limit switch?)
- Verify linearity of X-axis sensor and calibrate to position on sled
- Order Y and Z sensors, connect to sled
- Cobble existing code together for a user-friendly scan script
- Dedicated scopes for measurement setup (discussed with Bob)
- Determine whether we need to use hard stops, slip clutches, limit switches (software or "dead-man") or some combination thereof
- GUI

Thank you!